BAB 6 PENGUJIAN

Bab pengujian adalah bab yang berisi tentang pengujian yang dilakukan pada sistem. Tahap pengujian dilakukan untuk menguji hasil dari implementasi sistem yang telah dibuat. Pengujian tersebut berupa pengujian fungsional dan pengujian non fungsional. Dalam pengujian fungsional akan diuji dengan pengujian *black box* dan *white box.* Pengujian *white box* terdiri dari pengujian *unit*, sedangkan pengujian *black box* dengan pengujian validasi. Dalam pengujian non fungsional menggunakan pengujian *data integrity* dan *usability.*

6.1 Pengujian Fungsional

Dalam pengujian fungsional terdapat dua bagian yaitu pengujian *unit* dan pengujian validasi. Pengujian *unit* dilakukan untuk menguji setiap *method* pada fungsi utama yang diimplementasikan pada sistem. Sedangkan untuk pengujian validasi digunakan untuk menguji kebutuhan fungsional sistem apakah sudah sesuai dengan kebutuhan atau belum.

6.1.1 Pengujian validasi

Pengujian validasi digunakan untuk mengetahui apakah sistem yang dibuat telah benar dan sesuai dengan yang dibutuhkan. Pengujian validasi menggunakan metode pengujian *black box,* karena tidak memerlukan untuk mengetahui alur jalannya algoritma program dan lebih menekankan untuk menemukan validasi dari kebutuhan. Pada pengujian validasi dilakukan dua kali yaitu pengujian validasi iterasi pertama, dan pengujian validasi iterasi kedua. Hal ini dikarenakan setelah dilakukan pengujian validasi iterasi pertama terdapat penambahan kebutuhan fungsional dari *user*.

6.1.1.1 Pengujian validasi iterasi pertama

Pengujian validasi iterasi pertama adalah pengujian validasi yang dilakukan untuk menguji validasi dari kebutuhan fungsional iterasi pertama. Pengujian validasi iterasi pertama dapat dilihat pada tabel 6.1.

Tabel 6.1 Pengujian validasi iterasi pertama

No	Kebutuhan	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	*Login*	1. Aktor menekan link masuk. 2. Mengisikan form dengan username dan password yang benar. 3. Menekan tombol masuk.	Tulisan masuk akan berubah menjadi *dashboard.*	Tulisan masuk akan berubah menjadi *dashboard.*	Valid

		1. Aktor menekan link masuk. 2. Mengisikan form dengan username dan password yang salah. 3. Menekan tombol masuk.	Sistem akan menampilkan notifikasi "Invalid username or password".	Sistem akan menampilkan notifikasi "Invalid username or password".	Valid
2	Melihat Data Tilang *Non* User	Mengakses halaman awal *website.*	Menampilkan data tilang.	Menampilkan data tilang.	Valid
		Mengakses halaman awal *website* dan tidak ada data tilang.	Sistem menampilkan pesan "Data kosong".	Sistem menampilkan pesan "Data kosong".	Valid
3	*Filter* Data Tilang	1. Aktor memilih tanggal putusan sesuai tanggal putusan yang ada. 2. Aktor menekan tombol filter.	Sistem menampilkan data tilang.	Sistem menampilkan data tilang.	Valid
		1. Aktor mengisikan tanggal putusan kosong. 2. Aktor menekan tombol filter.	Sistem menampilkan pesan "pilih tanggal putusan".	Sistem menampilkan pesan "pilih tanggal putusan".	Valid
		1. Aktor mengisikan tanggal putusan 1 Januari 2017. 2. Aktor menekan tombol filter.	Menampilkan pesan data kosong.	Menampilkan pesan data kosong.	Valid
4	Mengganti Kata Sandi	1. Aktor melakukan autentikasi sebagai administrator. 2. Aktor menekan username. 3. Aktor menekan tombol ganti kata sandi. 4. Aktor mengisikan form. 5. Aktor menekan tombol simpan.	Sistem menampilkan pesan "Berhasil menggantikan kata sandi".	Sistem menampilkan pesan "Berhasil menggantikan kata sandi".	Valid
		1. Aktor melakukan autentikasi sebagai administrator.	Sistem menampilkan pesan "User dan	Sistem menampilkan pesan "User dan	Valid

		2. Aktor menekan username. 3. Aktor menekan tombol ganti kata sandi. 4. Aktor mengisikan form dengan data yang salah. 5. Aktor menekan tombol simpan.	kata sandi lama tidak sesuai".	kata sandi lama tidak sesuai".	
5	*Logout*	1. Aktor telah melakukan autentikasi kedalam sistem. 2. Aktor berada pada halaman dashboard. 3. Aktor menekan username dan menekan tombol keluar.	Sistem melakukan deautentifikasi, dan akan mengarahkan ke halaman utama *website.*	Sistem melakukan deautentifikasi, dan akan mengarahkan ke halaman utama *website.*	Valid
6	Menambah *User*	1. Aktor melakukan autentikasi sebagai administrator dan berada pada halaman konfigurasi. 2. Aktor menekan tombol tambah user. 3. Aktor mengisikan form. 4. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan "Berhasil menambahkan *user*".	Sistem akan menampilkan pesan "Berhasil menambahkan *user*".	Valid
		1. Aktor melakukan autentikasi sebagai administrator dan berada pada halaman konfigurasi. 2. Aktor menekan tombol tambah user. 3. Aktor mengisikan form dengan username yang telah dipakai. 4. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan "username telah dipakai".	Sistem akan menampilkan pesan "username telah dipakai".	Valid

7	Melihat Data *User*	1. Aktor melakukan autentikasi sebagai administrator. 2. Aktor mengakses halaman konfigurasi.	Sistem akan menampilkan data *user.*	Sistem akan menampilkan data *user.*	Valid
8	Menghapus *User*	1. Aktor melakukan autentikasi sebagai administrator dan berada pada halaman konfigurasi. 2. Aktor menekan tombol hapus pada user.	Sistem akan menampilkan pesan "Berhasil menghapus *user*".	Sistem akan menampilkan pesan "Berhasil menghapus *user*".	Valid
9	Melihat Data Tilang *User*	1. Aktor melakukan autentikasi sebagai administrator. 2. Aktor masuk ke halaman data tilang.	Sistem menampilkan data tilang.	Sistem menampilkan data tilang.	Valid
10	Mengedit Data Tilang	1. Aktor telah masuk sebagai administrator dan berada pada halaman data tilang. 2. Aktor menekan tombol edit. 3. Aktor melakukan perubahan data. 4. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan "Data tilang berhasil diperbaharui".	Sistem akan menampilkan pesan "Data tilang berhasil diperbaharui".	Valid
11	Melihat Statistik Data Tilang per Tahun	1. Aktor telah masuk sebagai operator. 2. Aktor masuk ke halaman dashboard.	Sistem menampilkan statistik dalam bentuk *chart*.	Sistem menampilkan statistik dalam bentuk *chart*.	Valid
12	Mengunduh Laporan Data Tilang Mingguan	1. Aktor telah masuk sebagai operator kepolisian dan berada pada halaman data tilang. 2. Aktor menekan tombol print. 3. Aktor memilih tanggal putusan.	Sistem akan mengunduh *file.*	Sistem akan mengunduh *file.*	Valid

		4. Aktor mengunduh file.			
13	Mengunduh *Template File* Data Tilang	1. Aktor telah masuk sebagai operator kepolisian dan berada pada halaman upload. 2. Aktor menekan tombol download template.	Sistem akan mengunduh *file*.	Sistem akan mengunduh *file*.	Valid
14	Mengunggah *File* Data Tilang	1. Aktor masuk sebagai operator kepolisian. 2. Aktor menekan tombol choose file. 3. Aktor memilih file excel yang diunggah. 4. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan "Data tilang berhasil disimpan".	Sistem akan menampilkan pesan "Data tilang berhasil disimpan".	Valid
		1. Aktor masuk sebagai operator kepolisian. 2. Aktor menekan tombol choose file. 3. Aktor memilih file excel yang tidak sesuai. 4. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan kesalahan.	Sistem akan menampilkan pesan kesalahan.	Valid
15	Melihat Data Tilang Mingguan Polisi	1. Aktor masuk sebagai operator kepolisian. 2. Aktor mengakses halaman upload data.	Sistem akan menampilkan data kosong.	Sistem akan menampilkan data kosong.	Valid
		1. Aktor masuk sebagai operator kepolisian. 2. Aktor mengakses halaman upload data, telah mengunggah data tilang da belum mengirimkan.	Sistem akan menampilkan data tilang dan pesan "Anda telah upload data untuk minggu ini, untuk upload ulang silahkan hapus semua data terlebih dahulu. jika sudah selesai klik opsi dan kirim untuk mengirimkan	Sistem akan menampilkan data tilang dan pesan "Anda telah upload data untuk minggu ini, untuk upload ulang silahkan hapus semua data terlebih dahulu. jika sudah selesai klik opsi dan kirim untuk mengirimkan	Valid

			data ke pengadilan.”.	data ke pengadilan.”.	
		1. Aktor masuk sebagai operator kepolisian. 2. Aktor mengakses halaman upload data, telah mengunggah data tilang dan telah mengirimkan ke pengadilan.	Sistem akan menampilkan pesan “data kosong” dan “Anda telah upload data untuk minggu ini, dan sudah dikirim.”.	Sistem akan menampilkan pesan “data kosong” dan “Anda telah upload data untuk minggu ini, dan sudah dikirim.”.	Valid
16	Mengirimkan Data Tilang	1. Aktor masuk sebagai operator kepolisian, berada pada halaman upload dan telah mengunggah file. 2. Aktor menekan tombol opsi, dan menekan tombol kirim data.	Sistem akan menampilkan pesan “Data berhasil dikirimkan ke pengadilan”	Sistem akan menampilkan pesan “Data berhasil dikirimkan ke pengadilan”	Valid
17	Menghapus Data Tilang Mingguan	1. Aktor masuk sebagai operaror kepolisian, berada pada halaman upload dan telah mengunggah file. 2. Aktor menekan tombol opsi, dan menekan tombol hapus semua.	Sistem akan menampilkan pesan “Berhasil menghapus semua data”.	Sistem akan menampilkan pesan “Berhasil menghapus semua data”.	Valid
18	Menambah Hakim	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman konfigurasi. 2. Aktor menekan tab hakim. 3. Aktor menekan tombol tambah hakim. 4. Aktor mengisi form. 5. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan “Berhasil menambahkan hakim”	Sistem akan menampilkan pesan “Berhasil menambahkan hakim”	Valid
		1. Aktor masuk sebagai operator pengadilan dan berada pada halaman konfigurasi.	Sistem akan menampilkan pesan “Kode hakim sudah dipakai”.	Sistem akan menampilkan pesan “Kode hakim sudah dipakai”.	Valid

		2. Aktor menekan tab hakim. 3. Aktor menekan tombol tambah hakim. 4. Aktor mengisi form dengan kode hakim yang sudah dipakai. 5. Aktor menekan tombol simpan.			
19	Melihat Data Hakim	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman konfigurasi. 2. Aktor menekan tab hakim.	Sistem akan menampilkan data hakim.	Sistem akan menampilkan data hakim.	Valid
20	Menghapus Hakim	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman konfigurasi. 2. Aktor menekan tab hakim. 3. Aktor menekan tombol dengan ikon tempat sampah.	Sistem akan menampilkan pesan “Berhasil menghapus hakim”.	Sistem akan menampilkan pesan “Berhasil menghapus hakim”.	Valid
21	Menambah Panitera	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman konfigurasi. 2. Aktor menekan tab panitera. 3. Aktor menekan tombol tambah panitera. 4. Aktor mengisi form. 5. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan “Berhasil menambahkan panitera”.	Sistem akan menampilkan pesan “Berhasil menambahkan panitera”.	Valid
		1. Aktor masuk sebagai operator pengadilan dan berada pada halaman konfigurasi. 2. Aktor menekan tab panitera.	Sistem akan menampilkan pesan “Kode panitera sudah dipakai”.	Sistem akan menampilkan pesan “Kode hakim sudah dipakai”.	Valid

		3. Aktor menekan tombol tambah panitera. 4. Aktor mengisi form dengan kode panitera yang sudah dipakai. 5. Aktor menekan tombol simpan.			
22	Melihat Data Panitera	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman konfigurasi. 2. Aktor menekan tab panitera.	Sistem akan menampilkan data panitera.	Sistem akan menampilkan data panitera.	Valid
23	Menghapus Panitera	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman konfigurasi. 2. Aktor menekan tab panitera. 3. Aktor menekan tombol dengan ikon tempat sampah.	Sistem akan menampilkan pesan “Berhasil menghapus panitera”.	Sistem akan menampilkan pesan “Berhasil menghapus panitera”.	Valid
24	Memasukkan Konfigurasi Persidangan	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman data minggu ini. 2. Aktor menekan tombol konfigurasi persidangan. 3. Aktor mengisikan form. 4. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan “Konfigurasi persidangan berhasil disimpan”.	Sistem akan menampilkan pesan “Konfigurasi persidangan berhasil disimpan”.	Valid
25	Melihat Data Tilang Mingguan Pengadilan	Aktor masuk sebagai operator pengadilan dan berada pada halaman data minggu ini.	Sistem akan menampilkan data tilang.	Sistem akan menampilkan data tilang.	Valid
		Aktor masuk sebagai operator pengadilan dan berada pada halaman data minggu ini	Sistem akan menampilkan pesan “Data kosong”.	Sistem akan menampilkan pesan “Data kosong”.	Valid

		dan tidak ada data tilang.			
26	Memasukkan Denda Tilang	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman minggu ini. 2. Aktor mengisikan denda tilang. 3. Aktor menekan tombol submit denda.	Sistem akan menampilkan pesan "Berhasil menginputkan denda".	Sistem akan menampilkan pesan "Berhasil menginputkan denda".	Valid
27	Konfirmasi Pembayaran Tilang	1. Aktor masuk sebagai operator kejaksaan dan berada pada halaman data tilang. 2. Aktor menekan tombol konfirmasi pembayaran. 3. Aktor memilih pembayaran ditempat. 4. Aktor menekan tombol simpan.	Sistem akan menampilkan pesan "Berhasil memasukkan tanggal pembayaran".	Sistem akan menampilkan pesan "Berhasil memasukkan tanggal pembayaran".	Valid

6.1.1.2 Pengujian validasi iterasi kedua

Pengujian validasi iterasi kedua dilakukan setelah tahap iterasi pertama selesai dilakukan dan terdapat penambahan kebutuhan fungsional dari *user*. Sehingga pada iterasi kedua terdapat pengujian validasi dari penambahan kebutuhan fungsional tersebut. Pengujian validasi iterasi kedua dapat dilihat pada tabel 6.2.

Tabel 6.2 Pengujian validasi iterasi kedua

No	Kebutuhan	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	Menampilkan Barang Bukti yang Belum Diambil	Aktor masuk sebagai operator kepolisian dan mengakses halaman barang bukti	Sistem akan menampilkan data tilang dengan barang bukti yang belum diambil.	Sistem akan menampilkan data tilang dengan barang bukti yang belum diambil.	Valid
2	Mengunduh Laporan Persidangan	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman data tilang.	Sistem akan mengunduh *file.*	Sistem akan mengunduh *file.*	Valid

		2. Aktor menekan tombol *print*. 3. Aktor memilih persidangan. 4. Aktor memilih tahun. 5. Aktor menekan tombol *download*.			
3	Menampilkan Total Perkara	Aktor masuk sebagai operator pengadilan dan berada pada halaman perkara per tahun.	Sistem menampilkan total perkara per tahun.	Sistem menampilkan total perkara per tahun.	Valid
4	Mengunduh Laporan Total Perkara	1. Aktor masuk sebagai operator pengadilan dan berada pada halaman perkara per tahun. 2. Aktor menekan tombol *print*. 3. Aktor memilih tahun. 4. Aktor menekan tombol *download*.	Sistem akan mengunduh *file.*	Sistem akan mengunduh *file.*	Valid

Berdasarkan hasil pengujian validasi tahap itrasi pertama dan kedua diketahui bahwa hasilnya adalah valid semua yang dapat dilihat pada tabel 6.1 dan tabel 6.2.

6.1.2 Pengujian *unit*

Pengujian *unit* adalah pengujian yang dilakukan untuk memastikan bahwa implementasi telah sesuai dengan yang diinginkan melalui pengujian algoritma. *Unit* yang dimaksud disini adalah *sebuah method* yang dibuat yang melakukan satu fungsi utama dan tidak memanggil *method* lain yang dibuat. Pada pengujian *unit* untuk *web service* menggunakan JUnit, dan untuk AngularJs menggunakan Jasmine. Pada pengujian *unit* hanya menguji *method-method* yang dijelaskan pada tahap implementasi.

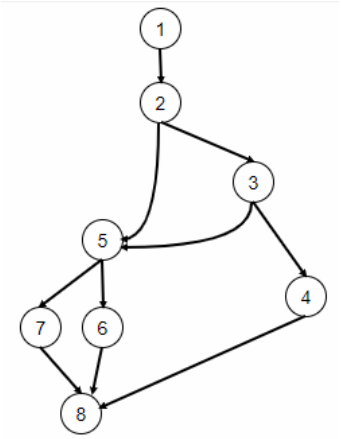
6.1.2.1 Pengujian *unit method* upload(file)

Pengujian *unit method* `upload(file)` digunakan untuk menguji *method* yang digunakan untuk mengunggah *file excel* data tilang. Gambar 6.1 merupakan kode program dari *method* `upload(file)` beserta node *flow graph*.

```
public void upload(MultipartFile file) throws Exception {
  ScheduleDto scheduleDto = scheduleService.findAll();                    (1)
  if (tigaPilarUtils.getCurrentLocaleDay()                                (2)
      .equalsIgnoreCase(scheduleDto.getPoliceDeliverySchedule())
      &&                                                                  (3)
      !fileService.isFileExist()) {
    int year = calendar.get(Calendar.YEAR);
    int week = calendar.get(Calendar.WEEK_OF_YEAR);
    String path = String.valueOf(year);                                   (4)
    String name = "Data_Tilang_" + year + "_" + week;
    String fileUploadPath = fileService.uploadFile(file, path, name);
    excelUtil.insertDataFromExcelFile(fileUploadPath);                    (5)
  } else if (fileService.isFileExist()) {
    throw new InvalidException("Anda telah mengirimkan berkas untuk minggu ini.");  (6)
  } else {                                                                (7)
    throw new InvalidException("Bukan hari pengiriman data tilang.");
  }                                                                       (8)
}
```

Gambar 6.1 Kode program *method* upload(file)

Berdasarkan hasil pemetaan *flow graph*, maka dibuat sebuah alur *flow graph* yang dapat dilihat pada gambar 6.2.



Gambar 6.2 *flow graph method* upload(file)

Setelah dibuat *flow graph*, selanjutnya adalah menentukan *cyclomatic complexity* untuk membuat kasus uji.

V(G) = E-N+2
= 10-8+2
= 4

Dari hasil *cyclomatic complexity,* dapat ditentukan jalur independen yaitu:

Jalur 1: 1-2-3-4-8
Jalur 2: 1-2-3-5-6-8
Jalur 3: 1-2-5-6-8
Jalur 4: 1-2-5-7-8

Sehingga dapat dibuat kasus uji seperti yang ditunjukkan pada tabel 6.3, kode pengujian ditunjukkan pada tabel 6.4, dan hasil pengujian menunjukkan bahwa pengujian ini valid seperti yang ditunjukkan gambar 6.3.

Tabel 6.3 Kasus uji *method* upload(file)

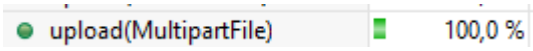
No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	upload(file)	Nilai `tigaPilarUtils.getCurrentLocaleDay().equalsIgnoreCase(scheduleDto.getPoliceDeliverySchedule())` bernilai *true* Nilai `!fileService.isFileExist()` bernilai *true*	Sistem akan memanggil *method* `insertDataFromExcelFile(excelFilePath)`	Sistem akan memanggil *method* `insertDataFromExcelFile(excelFilePath)`	Valid
2	upload(file)	Nilai `tigaPilarUtils.getCurrentLocaleDay().equalsIgnoreCase(scheduleDto.getPoliceDeliverySchedule())` bernilai *true* Nilai `!fileService.isFileExist()` bernilai *false*	Sistem akan melempar InvalidException	Sistem akan melempar InvalidException	Valid
3	upload(file)	Nilai `tigaPilarUtils.getCurrentLocaleDay().equalsIgnoreCase(scheduleDto.getPoliceDeliverySchedule())` bernilai *false* Nilai `!fileService.isFileExist()` bernilai *true*	Sistem akan melempar InvalidException	Sistem akan melempar InvalidException	Valid
4	upload(file)	Nilai `tigaPilarUtils.getCurrentLocaleDay().equalsIgnoreCase(scheduleDto.getPoliceDeliverySchedule())` bernilai *false* Nilai `!fileService.isFileExist()` bernilai *true*	Sistem akan melempar InvalidException	Sistem akan melempar InvalidException	Valid

Tabel 6.4 Pengujian *unit method* upload(file)

Pengujian unit method upload(file)

```
@Test
public void testUploadSuccess() throws Exception {
   ScheduleDto scheduleDto = new ScheduleDto("Senin", "Rabu");
   when(tigaPilarUtils.getCurrentLocaleDay()).thenReturn("Senin");
   when(scheduleService.findAll()).thenReturn(scheduleDto);
   when(fileService.isFileExist()).thenReturn(false);
   when(fileService.uploadFile(Mockito.any(), Mockito.any(),
   Mockito.any())).thenReturn("/file/upload/Data_Tilang_2017_1.xlsx");

   trafficTicketServiceImpl.upload(file);

   verify(fileService, times(1)).uploadFile(Mockito.any(),
   Mockito.any(), Mockito.any());
   verify(excelUtil,
   times(1)).insertDataFromExcelFile("/file/upload/Data_Tilang_2017_1.x
   lsx");
}

@Test(expected = InvalidException.class)
public void testUploadFailed1() throws Exception {
   ScheduleDto scheduleDto = new ScheduleDto("Senin", "Rabu");
   when(tigaPilarUtils.getCurrentLocaleDay()).thenReturn("Senin");
   when(scheduleService.findAll()).thenReturn(scheduleDto);
   when(fileService.isFileExist()).thenReturn(true);
   when(fileService.uploadFile(Mockito.any(), Mockito.any(),
   Mockito.any())).thenReturn("/file/upload/Data_Tilang_2017_1.xlsx");
   try {
      trafficTicketServiceImpl.upload(file);
   } catch (Exception e) {
      verify(fileService, times(0)).uploadFile(Mockito.any(),
      Mockito.any(), Mockito.any());
      verify(excelUtil,
      times(0)).insertDataFromExcelFile("/file/upload/Data_Tilang_2017_
      1.xlsx");
      throw e;
   }
}

@Test(expected = InvalidException.class)
public void testUploadFailed2() throws Exception {
     ScheduleDto scheduleDto = new ScheduleDto("Senin", "Rabu");
     when(tigaPilarUtils.getCurrentLocaleDay()).thenReturn("Selasa");
     when(scheduleService.findAll()).thenReturn(scheduleDto);
     when(fileService.isFileExist()).thenReturn(true);
     when(fileService.uploadFile(Mockito.any(), Mockito.any(),
     Mockito.any())).thenReturn("/file/upload/Data_Tilang_2017_1.xlsx");
     try {
        trafficTicketServiceImpl.upload(file);
     } catch (Exception e) {
        throw e;
     }
}

@Test(expected = InvalidException.class)
public void testUploadFailed3() throws Exception {
   ScheduleDto scheduleDto = new ScheduleDto("Senin", "Rabu");
   when(tigaPilarUtils.getCurrentLocaleDay()).thenReturn("Selasa");
   when(scheduleService.findAll()).thenReturn(scheduleDto);
   when(fileService.isFileExist()).thenReturn(false);
```

```
    when(fileService.uploadFile(Mockito.any(), Mockito.any(),
    Mockito.any())).thenReturn("/file/upload/Data_Tilang_2017_1.xlsx");
    try {
       trafficTicketServiceImpl.upload(file);
    } catch (Exception e) {
       verify(fileService, times(0)).uploadFile(Mockito.any(),
       Mockito.any(), Mockito.any());
       verify(excelUtil,
       times(0)).insertDataFromExcelFile("/file/upload/Data_Tilang_2017_
       1.xlsx");
       throw e;
    }
}
```



Gambar 6.3 Hasil pengujian *method* upload(file)

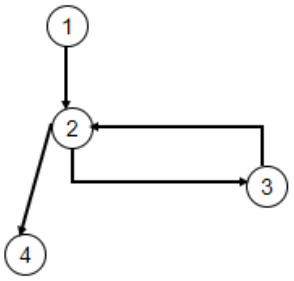
6.1.2.2 Pengujian *unit method* sendDataToCourt()

Pengujian *unit method* `sendDataToCourt()` digunakan untuk menguji *method* yang digunakan untuk mengirimkan data tilang yang telah diunggah kepolisian ke pengadilan. Gambar 6.4 merupakan kode program dari *method* `sendDataToCourt()` beserta *node flow graph*.

```
public void sendDataToCourt() throws Exception {
  CounterDto counterDto = counterService.findByYear();
  Institution institution =
      institutionRepository.findByInstitutionType(InstitutionType.PENGADILAN);
  List<TrafficTicket> trafficTickets = trafficTicketRepository.findWeeklyForPolice();
  long counter = counterDto.getCounter();

  for (TrafficTicket trafficTicket : trafficTickets) {
    trafficTicket.setCaseNumber(++counter);
    trafficTicket.setModifyForPolice(false);
    trafficTicket.setCourtCode(institution.getInstitutionWorkUnitCode());
  }
  counterDto.setCounter(counter);
  trafficTicketRepository.save(trafficTickets);
  counterService.save(counterDto);
}
```

Gambar 6.4 Kode program *method* sendDataToCourt()

Berdasarkan hasil pemetaan *flow graph*, maka dibuat sebuah alur *flow graph* yang dapat dilihat pada gambar 6.5.



Gambar 6.5 *Flow graph method* sendDataToCourt()

Setelah dibuat *flow graph*, selanjutnya adalah menentukan *cyclomatic complexity* untuk membuat kasus uji.

V(G) = E-N+2
= 4-4+2
= 2

Dari hasil cyclomatic *complexity*, dapat ditentukan jalur independen yaitu:

Jalur 1: 1-2-3-2-4
Jalur 2: 1-2-4

Sehingga dibuat kasus uji seperti yang ditunjukkan pada tabel 6.5, kode pengujian ditunjukkan pada tabel 6.6, dan hasil pengujian menunjukkan bahwa pengujian valid yang ditunjukkan oleh gambar 6.6.

Tabel 6.5 Kasus uji *method* sendDataToCourt()

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	sendDataToCourt()	Nilai variabel trafficTickets dengan panjang lebih dari 0.	Sistem melakukan *looping* sebanyak panjang trafficTickets.	Sistem melakukan *looping* sebanyak panjang trafficTickets.	Valid
2	sendDataToCourt()	Nilai variabel trafficTickets dengan panjang sama dengan 0.	Sistem tidak melakukan *looping*.	Sistem tidak melakukan *looping*.	Valid

Tabel 6.6 Pengujian *unit method* sendDataToCourt()

Pengujian unit method sendDataToCourt()

```
@Test
public void testSendDataToCourt1() throws Exception {
   counterDto = new CounterDto(Calendar.YEAR, 0L);
   institution = new Institution("Pengadilan Negeri",
   InstitutionType.PENGADILAN, "Imam", "123", 123);
   trafficTickets.add(trafficTicket);
   when(counterService.findByYear()).thenReturn(counterDto);
   when(institutionRepository.findByInstitutionType(InstitutionType.PEN
   GADILAN)).thenReturn(institution);
   when(trafficTicketRepository.findWeeklyForPolice()).thenReturn(traff
   icTickets);

   trafficTicketServiceImpl.sendDataToCourt();

   verify(counterService, times(1)).findByYear();
   verify(institutionRepository,
   times(1)).findByInstitutionType(InstitutionType.PENGADILAN);
   verify(trafficTicketRepository, times(1)).findWeeklyForPolice();
}

@Test
public void testSendDataToCourt2() throws Exception {
   counterDto = new CounterDto(Calendar.YEAR, 0L);
   institution = new Institution("Pengadilan Negeri",
   InstitutionType.PENGADILAN, "Imam", "123", 123);
```

```
    when(counterService.findByYear()).thenReturn(counterDto);
    when(institutionRepository.findByInstitutionType(InstitutionType.PEN
    GADILAN)).thenReturn(institution);
    when(trafficTicketRepository.findWeeklyForPolice()).thenReturn(traff
    icTickets);

    trafficTicketServiceImpl.sendDataToCourt();
    verify(counterService, times(1)).findByYear();
    verify(institutionRepository,
    times(1)).findByInstitutionType(InstitutionType.PENGADILAN);
    verify(trafficTicketRepository, times(1)).findWeeklyForPolice();
}
```

Gambar 6.6 Hasil pengujian *method* sendDataToCourt()

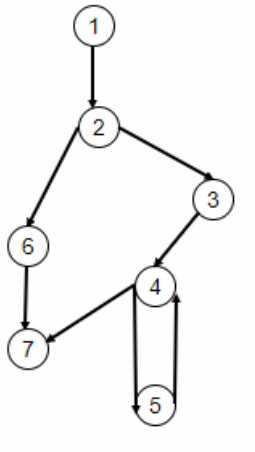
6.1.2.3 Pengujian *unit method* save(trialConfiguratioDto)

Pengujian *unit method* `save(trialConfigurationDto)` digunakan untuk menguji *method* yang digunakan untuk menyimpan konfigurasi persidangan. Gambar 6.7 merupakan kode program dari *method* `save(trialConfigurationDto)` berserta *node flow graph*.

```
public void save(TrialConfigurationDto trialConfigurationDto) throws Exception {
  TrialConfiguration trialConfiguration = trialConfigurationRepository
      .findByYearAndWeek(trialConfigurationDto.getYear(), calendar.get(Calendar.WEEK_OF_YEAR));
  trialConfigurationDto.setWeek(calendar.get(Calendar.WEEK_OF_YEAR));
  if (trialConfiguration == null) {
    trialConfiguration = trialConfigurationRepository
        .save(TrialConfigurationConverter.toEntity(trialConfigurationDto));

    for (TrialDataDto trialDataDto : trialConfigurationDto.getTrialDatas()) {
      TrialData trialData = TrialDataConverter.toEntity(trialDataDto);
      Judge judge = judgeRepository.findByCode(trialDataDto.getJudge());
      Clerks clerks = clerksRepository.findByCode(trialDataDto.getClerks());
      Presence presence = presenceRepository.findByPresence("Verstek");
      trialData.setJudge(judge);
      trialData.setClerks(clerks);
      trialData.setTrialConfiguration(trialConfiguration);

      trialDataRepository.save(trialData);
      trafficTicketRepository.updateTicketForCourt(trialConfigurationDto.getTrialDate(), judge,
          clerks, trialDataDto.getAdministrativeCost(), presence, "3 (tiga) hari kurungan",
          trialDataDto.getCaseNumberStart(), trialDataDto.getCaseNumberEnd(),
          trialConfigurationDto.getYear(), trialConfigurationDto.getWeek());
    }
  } else {
    throw new EntityExistsException(
        "Konfigurasi Persidangan untuk tahun: " + calendar.get(Calendar.YEAR) + " dan minggu: "
            + calendar.get(Calendar.WEEK_OF_YEAR) + " sudah ada.");
  }
}
```

Gambar 6.7 Kode program *method* save(trialConfigurationDto)

Berdasarkan hasil pemetaan *flow graph*, maka dibuat sebuah alur *flow graph* yang ditunjukkan oleh gambar 6.8.



Gambar 6.8 *Flow graph method* save(trialConfigurationDto)

Setelah dibuat *flow graph*, selanjutnya adalah menentukan *cyclomatic complexity* untuk membuat kasus uji.

V(G) = E-N+2

= 8-7+2

= 3

Dari hasil *cyclomatic complexity,* dapat ditentukan jalur independen yaitu:

Jalur 1: 1-2-3-4-5-4-7

Jalur 2: 1-2-3-4-7

Jalur 3: 1-2-6-7

Sehingga dapat dibuat kasus uji seperti yang ditunjukkan pada tabel 6.7, kode pengujian ditunjukkan pada tabel 6.8, dan hasil pengujian menunjukkan bahwa pengujian ini valid seperti yang ditunjukkan gambar 6.9.

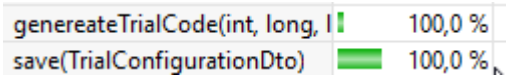
Tabel 6.7 Kasus uji *method* save(trialConfigurationDto)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	save(trialConfigurationDto)	Nilai variabel trialConfiguration sama dengan null. Panjang *list* dari pemanggilan *method* trialConfigurationDto lebih dari 0.	Menjalankan *method* save(trialConfiguration) save(trialData) updateTicketForCourt(verdictDate, judge, clerks, administrativeCost, presence, subsider, caseNumberStart, caseNumberEnd, year, week)	Menjalankan *method* save(trialConfiguration) save(trialData) updateTicketForCourt(verdictDate, judge, clerks, administrativeCost, presence, subsider, caseNumberStart, caseNumberEnd, year, week)	Valid
2	save(trialconfigurationDto)	Nilai variabel trialConfiguration sama dengan null. Panjang *list* dari pemanggilan *method* trialConfigurationDto sama dengan 0.	Menjalankan *method* save(trialConfiguration)	Menjalankan *method* save(trialConfiguration)	Valid
3	Save(trialConfigurationDto)	nilai variabel trialConfiguration selain null.	Sistem melempar EntityExistException.	Sistem melempar EntityExistException.	Valid

Tabel 6.8 Pengujian *unit method* save(trialConfigurationDto)

Pengujian unit method save(trialConfigurationDto)

```
@Test
public void testSaveSuccess() throws Exception {
     trialConfiguration =
     TrialConfigurationConverter.toEntity(trialConfigurationDto);
     when(trialConfigurationRepository.findByYearAndWeek(Mockito.anyInt(
     ), Mockito.anyInt())).thenReturn(null);
     when(trialConfigurationRepository
     .save(TrialConfigurationConverter.toEntity(trialConfigurationDto)))
     .thenReturn(trialConfiguration);
     trialConfigurationServiceImpl.save(trialConfigurationDto);
     verify(trialConfigurationRepository,
     times(1)).findByYearAndWeek(Mockito.anyInt(),Mockito.anyInt());
     verify(judgeRepository, times(1)).findByCode(Mockito.anyString());
     verify(clerksRepository, times(1)).findByCode(Mockito.anyString());
     verify(presenceRepository,
     times(1)).findByPresence(Mockito.anyString());
}
@Test
public void testSaveEmptyTrialData() throws Exception {
     trialDataDtos.clear();
     trialConfigurationDto = new TrialConfigurationDto("1", new Date(),
     new Date(), new Date(), new Date(), new Date(), trialDataDtos,
     Calendar.YEAR, Calendar.WEEK_OF_YEAR);
     when(trialConfigurationRepository.findByYearAndWeek(Mockito.anyInt(
     ), Mockito.anyInt())).thenReturn(null);
     when(judgeRepository.findByCode(Mockito.anyString())).thenReturn(ne
     w Judge());
     when(clerksRepository.findByCode(Mockito.anyString())).thenReturn(n
     ew Clerks());
     when(presenceRepository.findByPresence(Mockito.anyString())).thenRe
     turn(new Presence());
     trialConfigurationServiceImpl.save(trialConfigurationDto);
     verify(trialConfigurationRepository,
     times(1)).findByYearAndWeek(Mockito.anyInt(),Mockito.anyInt());
     verify(judgeRepository, times(0)).findByCode(Mockito.anyString());
     verify(clerksRepository, times(0)).findByCode(Mockito.anyString());
     verify(presenceRepository,
     times(0)).findByPresence(Mockito.anyString());
}
@Test(expected = EntityExistsException.class)
public void testSaveFailed() throws Exception {
     when(trialConfigurationRepository.findByYearAndWeek(Mockito.anyInt(
     ), Mockito.anyInt())).thenReturn(trialConfiguration);
     try {
        trialConfigurationServiceImpl.save(trialConfigurationDto);
     } catch (Exception e) {
        verify(trialConfigurationRepository,
        times(1)).findByYearAndWeek(Mockito.anyInt(),Mockito.anyInt());
        verify(judgeRepository,
        times(0)).findByCode(Mockito.anyString());
        verify(clerksRepository,
        times(0)).findByCode(Mockito.anyString());
        verify(presenceRepository,
        times(0)).findByPresence(Mockito.anyString());
        throw e;
     }
}
```



Gambar 6.9 Hasil pengujian *method* save(trialConfigurationDto)

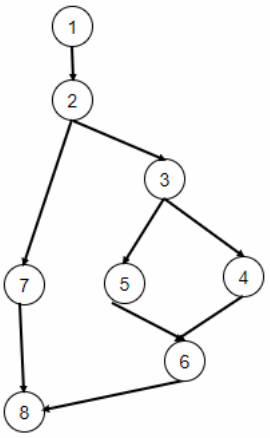
6.1.2.4 Pengujian *unit method* inputCost(ticketRegisterNumber, cost)

Pengujian *unit method* `inputCost(ticketRegisterNumber, cost)` digunakan untuk menguji *method* yang digunakan untuk memasukkan denda tilang. Gambar 6.10 merupakan kode program dari *method* `inputCost(ticketRegisterNumber, cost)` berserta *node flow graph*.

```
public void inputCost(String ticketRegisterNumber, double cost) throws Exception {
  TrafficTicket trafficTicket = trafficTicketRepository
      .findByTicketRegisterNumberAndYear(ticketRegisterNumber, calendar.get(Calendar.YEAR));
  if (trafficTicket != null) {
    if (trafficTicket.getPaymentDate() != null) {
      double changeMoney =
          trafficTicket.getDepositMoney() - (cost + trafficTicket.getAdministrativeCost());
      trafficTicket.setCost(cost);
      trafficTicket.setChangeMoney(changeMoney);
    } else {
      trafficTicket.setCost(cost);
    }
    trafficTicketRepository.save(trafficTicket);
  } else {
    throw new EntityNotFoundException("Data tilang tidak ditemukan.");
  }
}
```

Gambar 6.10 Kode program *method* inputCost(ticketRegisterNumber, cost)

Berdasarkan hasil pemetaan *flow graph*, maka dibuat sebuah alur *flow graph* berdasarkan gambar 6.10 yang dapat dilihat pada gambar 6.11.



Gambar 6.11 *Flow graph method* inputCost(ticketRegisterNumber, cost)

Setelah dibuat *flow graph*, selanjutnya adalah menentukan *cyclomatic complexity* untuk membuat kasus uji.

V(G) = E-N+2
= 9-8+2
= 3

Dari hasil *cyclomatic complexity*, dapat ditentukan jalur independen yaitu:

Jalur 1: 1-2-3-4-6-8
Jalur 2: 1-2-3-5-6-8

Jalur 3: 1-2-7-8

Sehingga dapat dibuat kasus uji seperti yang ditunjukkan tabel 6.9, kode pengujian ditunjukkan pada tabel 6.10, dan hasil pengujian menunjukkan bahwa pengujian ini valid yang ditunjukkan oleh gambar 6.12.

Tabel 6.9 Kasus uji *method* inputCost(ticketRegisterNumber, cost)

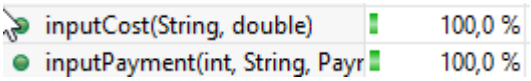
No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	inputCost(ticketRegisterNumber, cost)	Nilai variabel trafficTicket tidak sama dengan null. Nilai dari pemanggilan *method* getPaymentDate() tidak sama dengan null.	Melakukan kalkulasi uang kembalian yang disimpan pada variabel changeMoney.	Melakukan kalkulasi uang kembalian yang disimpan pada variabel changeMoney.	Valid
2	inputCost(ticketRegisterNumber, cost)	Nilai variabel trafficTicket tidak sama dengan null. Nilai dari pemanggilan *method* getPaymentDate selain tidak sama dengan null.	Mengeset nilai variabel cost pada variabel trafficTicket.	Mengeset nilai variabel cost pada variabel trafficTicket.	Valid
3	inputCost(ticketRegisterNumber, cost)	Nilai variabel traficTicket selain tidak sama dengan null	Sistem melempar EntityNotFoundException.	Sistem melempar EntityNotFoundException.	Valid

Tabel 6.10 Pengujian *unit method* inputCost(ticketRegisterNumber, cost)

Pengujian *unit method* inputCost(ticketRegisterNumber, cost)

```
@Test
public void testInputCostSuccessHavePaymentDate() throws Exception {
    trafficTicket.setPaymentDate(new Date());
    trafficTicket.setDepositMoney(80000);
    trafficTicket.setAdministrativeCost(1000);

    when(trafficTicketRepository.findByTicketRegisterNumberAndYear(Mock
    ito.anyString(), Mockito.anyInt())).thenReturn(trafficTicket);

    trafficTicketServiceImpl.inputCost("1", 70000);

    verify(trafficTicketRepository,
    times(1)).findByTicketRegisterNumberAndYear(Mockito.anyString(),
    Mockito.anyInt());
}

@Test
public void testInputCostSuccessDontHavePaymentDate() throws Exception
{
```

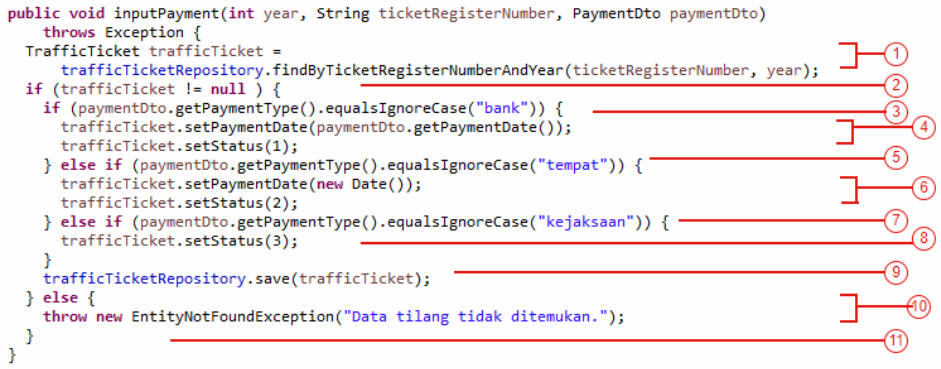
```
    trafficTicket.setDepositMoney(80000);
    trafficTicket.setAdministrativeCost(1000);

    when(trafficTicketRepository.findByTicketRegisterNumberAndYear(Mock
    ito.anyString(), Mockito.anyInt())).thenReturn(trafficTicket);

    trafficTicketServiceImpl.inputCost("1", 70000);

    verify(trafficTicketRepository,
    times(1)).findByTicketRegisterNumberAndYear(Mockito.anyString(),
    Mockito.anyInt());
}

@Test(expected = EntityNotFoundException.class)
public void testInputCostFailed() throws Exception {
    when(trafficTicketRepository.findByTicketRegisterNumberAndYear(Mock
    ito.anyString(), Mockito.anyInt())).thenReturn(null);

    try {
        trafficTicketServiceImpl.inputCost("1", 70000);
    } catch (Exception e) {
        verify(trafficTicketRepository, times(1))
        .findByTicketRegisterNumberAndYear(Mockito.anyString(),
        Mockito.anyInt());
        throw e;
    }
}
```



Gambar 6.12 Hasil pengujian *method* inputCost(ticketRegisterNumber, cost)

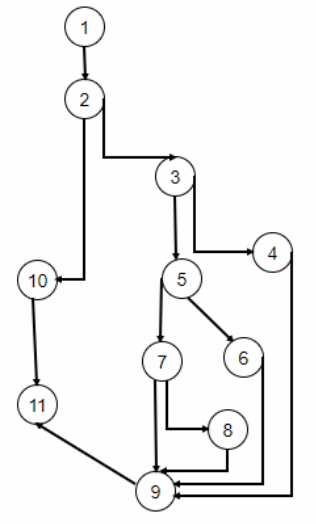
6.1.2.5 Pengujian *unit method* inputPayment(year, ticketRegisterNumber, paymentDto)

Pengujian *unit method* `inputPayment(year, ticketRegisterNumber, paymentDto)` digunakan untuk menguji *method* yang digunakan untuk melakukan konfirmasi pembayaran tilang. Gambar 6.13. merupakan kode program dari *method* `inputPayment(year, ticketRegisterNumber, paymentDto)` beserta *node flow graph*.



Gambar 6.13 Kode program *method* inputPayment(year, ticketRegisterNumber, paymentDto)

Berdasarkan hasil pemetaan *flow graph*, maka dibuat sebuah alur *flow graph* yang dapat dilihat pada gambar 6.14.



Gambar 6.14 *Flow graph method* inputPayment(year, ticketRegisterNumber, paymentDto)

Setelah dibuat *flow grap,* selanjutnya adalah menentukan *cyclomatic complexity* untuk membuat kasus uji.

V(G) = E-N+2

= 14-11+2

= 5

Dari hasil *cylomatic complexity*, dapt ditentukan jalur independen yaitu:

Jalur 1: 1-2-3-4-9-11

Jalur 2: 1-2-3-5-6-9-11

Jalur 3: 1-2-3-5-7-8-9-11

Jalur 4: 1-2-3-5-7-9-11

Jalur 5: 1-2-10-11

Sehingga dapat dibuat kasus uji seperti yang ditunjukkan pada tabel 6.11, kode pengujian ditunjukkan pada tabel 6.12, dan hasil pengujin menunjukkan bahwa pengujian valid seperti yang ditunjukkan gambar 6.15.

Tabel 6.11 Kasus uji *method* inputPayment(year, ticketRegisterNumber, paymentDto)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	inputPayment(year, ticketRegisterNumber, payment)	Nilai trafficTicket tidak sama dengan null. Nilai paymentType sama dengan bank.	Data disimpan dengan paymentDate dari nilai variable paymentDate dan status 1.	Data disimpan dengan paymentDate dari nilai variable paymentDate dan status 1.	Valid
2	inputPayment(year, ticketRegisterNumber, payment)	Nilai trafficTicket tidak sama dengan null. Nilai paymentType sama dengan tempat.	Data disimpan dengan paymentDate tanggal hari ini dan status 2.	Data disimpan dengan paymentDate tanggal hari ini dan status 2.	Valid
3	inputPayment(year, ticketRegisterNumber, payment)	Nilai trafficTicket tidak sama dengan null. Nilai paymentType sama dengan kejaksaan.	Data disimpan dengan status 3.	Data disimpan dengan status 3.	Valid
4	inputPayment(year, ticketRegisterNumber, payment)	Nilai trafficTicket tidak sama dengan null. Nilai paymentType selain bank, tempat, kejaksaan.	Data disimpan.	Data disimpan	Valid
5	inputPayment(year, ticketRegisterNumber, payment)	Nilai trafficTicket selain tidak sama dengan null	Melempar EntityNotFoundException.	Melempar EntityNotFoundException.	Valid

Tabel 6.12 Pengujian *unit method* inputPayment(year, ticketRegisterNumber, paymentDto)

Pengujian *unit method* inputPayment(year, ticketRegisterNumber, paymentDto)

```
@Test
public void testInputPaymentSuccess1() throws Exception {
    PaymentDto paymentDto = new PaymentDto("bank", new Date());
    when(trafficTicketRepository.findByTicketRegisterNumberAndYear(Mock
    ito.anyString(), Mockito.anyInt())).thenReturn(trafficTicket);
    trafficTicketServiceImpl.inputPayment(Calendar.YEAR, "1",
    paymentDto);
    verify(trafficTicketRepository,
    times(1)).findByTicketRegisterNumberAndYear(Mockito.anyString(),Moc
    kito.anyInt());
}
@Test
public void testInputPaymentSuccess2() throws Exception {
    PaymentDto paymentDto = new PaymentDto("tempat", new Date());
    when(trafficTicketRepository.findByTicketRegisterNumberAndYear(Mock
    ito.anyString(), Mockito.anyInt())).thenReturn(trafficTicket);
    trafficTicketServiceImpl.inputPayment(Calendar.YEAR, "1",
    paymentDto);
    verify(trafficTicketRepository,
    times(1)).findByTicketRegisterNumberAndYear(Mockito.anyString(),
    Mockito.anyInt());
}
@Test
public void testInputPaymentSuccess3() throws Exception {
    PaymentDto paymentDto = new PaymentDto("kejaksaan", new Date());
    when(trafficTicketRepository.findByTicketRegisterNumberAndYear(Mock
    ito.anyString(), Mockito.anyInt())).thenReturn(trafficTicket);
    trafficTicketServiceImpl.inputPayment(Calendar.YEAR, "1",
    paymentDto);
    verify(trafficTicketRepository,
    times(1)).findByTicketRegisterNumberAndYear(Mockito.anyString(),
    Mockito.anyInt());
}
@Test
public void testInputPaymentSuccess4() throws Exception {
    trafficTicket.setPaymentDate(new Date());
    PaymentDto paymentDto = new PaymentDto("", new Date());
    when(trafficTicketRepository.findByTicketRegisterNumberAndYear(Mock
    ito.anyString(),
    Mockito.anyInt())).thenReturn(trafficTicket);
    trafficTicketServiceImpl.inputPayment(Calendar.YEAR, "1",
    paymentDto);
}
@Test(expected = EntityNotFoundException.class)
public void testInputPaymentFailed() throws Exception {
    PaymentDto paymentDto = new PaymentDto("bank", new Date());
    when(trafficTicketRepository.findByTicketRegisterNumberAndYear(Mock
    ito.anyString(), Mockito.anyInt())).thenReturn(null);
    try {
        trafficTicketServiceImpl.inputPayment(Calendar.YEAR, "1",
        paymentDto);
    } catch (Exception e) {
        verify(trafficTicketRepository, times(1))
        .findByTicketRegisterNumberAndYear(Mockito.anyString(),
        Mockito.anyInt());
        throw e;
    }
  }
```

getCaseReportByYear(int)		100,0 %
inputCost(String, double)		100,0 %
inputPayment(int, String, Payr		100,0 %

Gambar 6.15 Hasil pengujian *method* inputPayment(year, ticketRegisterNumber, paymentDto)

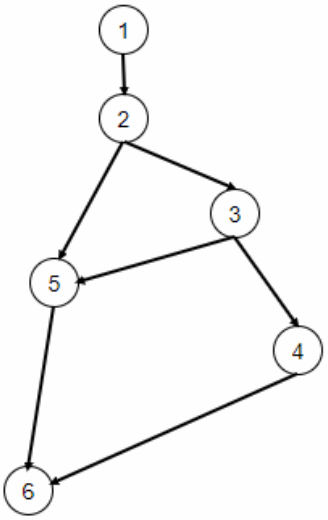
6.1.2.6 Pengujian *unit method* findAvailableEvidence(year, month) iterasi kedua

Pengujian *unit method* `findAvailableEvidence(year, month)` dilakukan pada iterasi kedua, *method* ini digunakan untuk menampilkan data tilang dengan barang bukti yang belum diambil atau masih di kejaksaan. Gambar 6.16 merupakan kode program dari *method* `findAvailableEvidence(year, month)` beserta *node flow graph*.

```
public List<TrafficTicket> findAvailableEvidence(int year, int month) throws Exception {
  List<TrafficTicket> trafficTickets = new ArrayList<>();
  List<Integer> status = new ArrayList<>();
  status.add(0);
  status.add(3);
  if (year == 0
      &&
      month == 0) {
    trafficTickets = trafficTicketRepository.findByYearAndMonthAndStatusIn(
        calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH) + 1, status);
  } else {
    trafficTickets = trafficTicketRepository.findByYearAndMonthAndStatusIn(year, month, status);
  }
  return trafficTickets;
}
```

Gambar 6.16 Kode program *method* findAvailableEvidence(year, month) iterasi kedua

Berdasarkan hasil pemetaan *flow graph,* maka dibuat sebuah alur *flow graph* yang ditunjukkan oleh gambar 6.17.



Gambar 6.17 *Flow graph method* findAvailableEvidence(year, month) iterasi kedua

Setelah dibuat *flow graph*, selanjutnya adalah menentukan *cyclomaic complexity* untuk membuat kasus uji.

V(G) = E-N+2
= 7-6+2
= 3

Dari hasil *cyclomatic complexity*, dapat ditentukan jalur independen yaitu:

Jalur 1: 1-2-3-4-6
Jalur 2: 1-2-3-5-6
Jalur 3: 1-2-5-6

Sehingga dapat dibuat kasus uji yang ditunjukkan pada tabel 6.13, kode pengujian ditunjukkan pada tabel 6.14, dan hasil pengujian ditunjukkan oleh gambar 6.18.

Tabel 6.13 Kasus uji *method* findAvailableEvidence(year, month) iterasi kedua

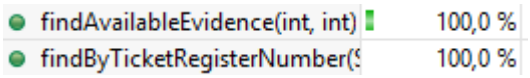
No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	findAvailableEvidence(year, month)	Nilai variabel *year* dan *month* sama dengan 0	Mengembalikan nilai variabel trafficTickets.	Mengembalikan nilai variabel trafficTickets.	Valid
2	findAvailableEvidence(year, month)	Nilai variabel *year* sama dengan 0 dan *month* selain sama dengan 0	Mengembalikan nilai variabel trafficTickets.	Mengembalikan nilai variabel trafficTickets.	Valid
3	findAvailableEvidence(year, month)	Nilai variabel *year* dan *month* selain sama dengan 0	Mengembalikan nilai variabel trafficTickets.	Mengembalikan nilai variabel trafficTickets.	Valid

Tabel 6.14 Pengujian *unit method* findAvailableEvidence(year, month) iterasi kedua

Pengujian *unit method* findAvailableEvidence(year, month)

```
@Test
public void testFindAvailableEvidence1() throws Exception {
    trafficTickets.add(trafficTicket);

    when(trafficTicketRepository.findByYearAndMonthAndStatusIn(Mockito.
    anyInt(), Mockito.anyInt(),
    Mockito.anyList())).thenReturn(trafficTickets);

    trafficTicketServiceImpl.findAvailableEvidence(0, 0);

    verify(trafficTicketRepository,
    times(1)).findByYearAndMonthAndStatusIn(Mockito.anyInt(),
    Mockito.anyInt(), Mockito.anyList());
}

@Test
public void testFindAvailableEvidence2() throws Exception {
    trafficTickets.add(trafficTicket);
```

```
    when(trafficTicketRepository.findByYearAndMonthAndStatusIn(Mockito.
    anyInt(), Mockito.anyInt(),
    Mockito.anyList())).thenReturn(trafficTickets);
        trafficTicketServiceImpl.findAvailableEvidence(Calendar.YEAR,
    Calendar.MONTH);
    verify(trafficTicketRepository,
    times(1)).findByYearAndMonthAndStatusIn(Mockito.anyInt(),
    Mockito.anyInt(), Mockito.anyList());
}

@Test
public void testFindAvailableEvidence3() throws Exception {
    trafficTickets.add(trafficTicket);

    when(trafficTicketRepository.findByYearAndMonthAndStatusIn(Mockito.
    anyInt(), Mockito.anyInt(),
    Mockito.anyList())).thenReturn(trafficTickets);

    trafficTicketServiceImpl.findAvailableEvidence(0, Calendar.MONTH);
}
```



Gambar 6.18 Hasil pengujian *method* findAvailableEvidence(year, month) iterasi kedua

6.1.2.7 Pengujian *unit method* UploadFactory.post(file)

Pengujian *unit method* `UploadFactory.post(file)` digunakan untuk menguji *service* pada AngularJs yang digunakan untuk mengakses ke *web service* mengunggah *file excel* data tilang. Pada pengujian ini dilakukan dengan menggunakan Jasmine yang dapat dilihat pada tabel 6.17. Kasus uji untuk pengujian ini dapat dilihat pada tabel 6.16, dan hasil dari pengujian dapat dilihat pada gambar 6.19.

Tabel 6.15 Kasus uji *method* UploadFactory.post(file)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	UploadFactory.post(file)	Mengakses URI /TigaPilar/api/ticket/upload dengan HTTP method POST	Mengembalikan *response* dengan nilai *message* Data tilang berhasil disimpan.	Mengembalikan *response* dengan nilai *message* Data tilang berhasil disimpan.	Valid

Tabel 6.16 Pengujian *unit method* UploadFactory.post(file)

Pengujian *unit method* UploadFactory.post(file)

`it('should return Data tilang berhasil disimpan', function(){`
`  var mockFile = {file:[{"name":"file.xls", "size":1018,`
`  "type":"application/vnd.ms-excel"}]};`

```
    httpBackend.expect('POST',
    '/TigaPilar/api/ticket/upload?requestId=' +
    UtilService.get()).respond(200, response1);

    var responseData = null;
    var result = UploadFactory.post(mockFile).then(function (data) {
      responseData = data;
    });
    httpBackend.flush();

    expect(responseData.message).toBe("Data tilang berhasil disimpan");
});
```

```
Kepolisian - Upload Service
  should return Data tilang berhasil disimpan
  should return Data berhasil dikirimkan ke pengadilan
```

Gambar 6.19 Hasil pengujian *method* UploadFactory.post(file)

6.1.2.8 Pengujian *unit method* SendDataFactory.put(request)

Pengujian *unit method* `SendDataFactory.put(request)` digunakan untuk menguji *service* pada AngularJs yang digunakan untuk mengakses ke *web service* mengirimkan data tilang. Pada pengujian ini dilakukan dengan menggunakan Jasmine yang dapat dilihat pada tabel 6.18. Kasus uji pada pengujian ini dapat dilihat pada tabel 6.17, dan hasil dari pengujian dapat dilihat pada gambar 6.20.

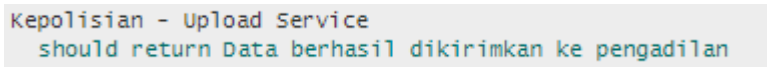
Tabel 6.17 Kasus uji *method* SendDataFactory.put(request)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	SendDataFactory.put(request)	Mengakses URI /TigaPilar/api/ticket/send dengan HTTP method PUT	Mengembalikan *response* dengan nilai *message* Data berhasil dikirimkan ke pangadilan.	Mengembalikan *response* dengan nilai *message* Data berhasil dikirimkan ke pangadilan.	Valid

Tabel 6.18 Pengujian *unit method* SendDataFactory.put(request)

Pengujian *unit method* SendDataFactory.post(request)

`it('should return Data berhasil dikirimkan ke pengadilan', function() {`
`  httpBackend.expect('PUT', '/TigaPilar/api/ticket/send?requestId=' +`
`  UtilService.get(), request).respond(200, response);`
`  var result = SendDataFactory.put(request);`
`  httpBackend.flush();`
`  expect(result.message).toBe("Data berhasil dikirimkan ke`
`  pengadilan");`
`});`



Gambar 6.20 Hasil pengujian *method* SendDataFactory.put(request)

6.1.2.9 Pengujian *unit method* SaveTrialFactory.post(request)

Pengujian *unit method* `SaveTrialFactory.post(request)` digunakan untuk menguji *service* pada AngularJs yang digunakan untuk mengakses ke *web service* yang digunakan untuk menyimpan konfigurasi persidangan. Pada pengujian ini dilakukan dengan menggunakan Jasmine yang dapat dilihat pada tabel 6.21. Kasus uji pada pengujian ini dapat dilihat pada tabel 6.19, dan hasil dari pengujian dapat dilihat pada gambar 6.21.

Tabel 6.19 Kasus uji *method* SaveTrialFactory.post(request)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	SaveTrialFactory.post(request)	Mengakses URI /TigaPilar/api/trial dengan HTTP method POST dan *request* yang ditunjukkan tabel 6.20	Mengembalikan *response* dengan nilai *message* Konfigurasi persidangan berhasil disimpan.	Mengembalikan *response* dengan nilai *message* Konfigurasi persidangan berhasil disimpan.	Valid

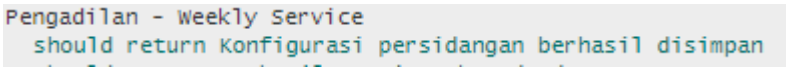
Tabel 6.20 *Request* kasus uji *method* SaveTrialFactory.post(request)

```
var request1 = {
    "clerksDeterminationDate": "2017-12-22T15:36:51.050Z",
    "id": "string",
    "judgeDeterminationDate": "2017-12-22T15:36:51.050Z",
    "registerDate": "2017-12-22T15:36:51.050Z",
    "trialDatas": [{
      "administrativeCost": 0,
      "caseNumberEnd": 0,
      "caseNumberStart": 0,
      "clerks": "string",
      "judge": "string",
      "room": "string"
    }],
    "trialDate": "2017-12-22T15:36:51.050Z",
    "trialDeterminationDate": "2017-12-22T15:36:51.050Z",
    "week": 0,
    "year": 0
  };
```

Tabel 6.21 Pengujian *unit method* SaveTrialFactory.post(request)

Pengujian *unit method* SaveTrialFactory.post(request)
`it('should return Konfigurasi persidangan berhasil disimpan', function() {`

```
    httpBackend.expect('POST', '/TigaPilar/api/trial?requestId=' +
    UtilService.get(), request1).respond(200, response1);
    var result = SaveTrialFactory.post(request1);
    httpBackend.flush();
    expect(result.message).toBe("Konfigurasi persidangan berhasil
    disimpan");
});
```



Gambar 6.21 Hasil pengujian *method* SaveTrialFactory.post(request)

6.1.2.10 Pengujian *unit method* InputCostFactory.post(request)

Pengujian *unit method* `InputCostFactory.post(request)` digunakan untuk menguji *service* pada AngularJs yang digunakan untuk mengakses ke *web service* memasukkan denda tilang. Pada pengujian ini dilakukan dengan menggunakan Jasmine yang dapat dilihat pada tabel 6.24. Kasus uji pada pengujian ini dapat dilihat pada tabel 6.22, dan hasil dari pengujian dapat dilihat pada gambar 6.23.

Tabel 6.22 Kasus uji *method* InputCostFactory.post(request)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	InputCostFactory.post(request)	Mengakses URI /TigaPilar/api/ticket/input-cost dengan HTTP method POST dan *request* yang ditunjukkan tabel 6.23	Mengembalikan *response* dengan *message* Berhasil menginputkan denda.	Mengembalikan *response* dengan *message* Berhasil menginputkan denda.	Valid

Tabel 6.23 *Request* kasus uji *method* InputCostFactory.post(request)

```
  var request2 = {
    ticketRegisterNumber: "A201",
    cost: 50000
  };
```

Tabel 6.24 Pengujian *unit method* InputCostFactory.post(request)

Pengujian *unit method* InputCostFactory.post(request)

`it('should return Berhasil menginputkan denda', function() {`
`  httpBackend.expect('POST', '/TigaPilar/api/ticket/input-cost?cost='`
`  + request2.cost + '&requestId='+UtilService.get()+`
`  '&ticketRegisterNumber='+request2.ticketRegisterNumber).respond(200,`
`  response2);`
 	
`  var result = InputCostFactory.post(request2);`
`  httpBackend.flush();`
`  expect(result.message).toBe("Berhasil menginputkan denda");`

```
});
```

```
Pengadilan - Weekly Service
  should return Konfigurasi persidangan berhasil disimpan
  should return Berhasil menginputkan denda
```

Gambar 6.22 Hasil pengujian *method* InputCostFactory.post(request)

6.1.2.11 Pengujian *unit method* InputPaymentFactory.put(request)

Pengujian *unit method* `InputPaymentFactory.put(request)` digunakan untuk menguji *service* pada AngularJs yang digunakan untuk mengakses ke *web service*. Pada pengujian ini dilakukan dengan menggunakan Jasmine yang dapat dilihat pada tabel 6.27. Kasus uji pada pengujian ini dapat dilihat pada tabel 6.25, dan hasil dari pengujian dapat dilihat pada gambar 6.24.

Tabel 6.25 Kasus uji *method* InputPaymentFactory.put(request)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	InputPaymentFactory.post(request)	Mengakses URI /TigaPilar/api/ticket dengan HTTP method PUT dan *request* yang ditunjukkan tabel 6.26	Mengembalikan *response* dengan *message* Berhasil memasukkan tanggal pembayaran.	Mengembalikan *response* dengan *message* Berhasil memasukkan tanggal pembayaran.	Valid

Tabel 6.26 *Request* kasus uji *method* InputPaymentFactory.put(request)

```
var request = {
    "paymentDate": "2017-12-22T15:36:51.050Z",
    "paymentType": "tempat"
};
```

Tabel 6.27 Pengujian *unit method* InputPaymentFactory.put(request)

Pengujian *unit method* InputPaymentFactory.put(request)

`it('should return Berhasil memasukkan tanggal pembayaran', function() {`
`  httpBackend.expect('PUT', '/TigaPilar/api/ticket?requestId=' +`
`  UtilService.get(), request).respond(200, response);`
`  var result = InputPaymentFactory.put(request);`
`  httpBackend.flush();`
`  expect(result.message).toBe("Berhasil memasukkan tanggal`
`  pembayaran");`
`});`

```
Kejaksaan - Data Service
  should return Berhasil memasukkan tanggal pembayaran
```

Gambar 6.23 Hasil pengujian *method* InputPaymentFactory.put(request)

6.1.2.12 Pengujian *unit method* GetAvailableEvidenceFactory.get(request) iterasi kedua

Pengujian *unit method* `GetAvailableEvidenceFactory.get(request)` digunakan untuk menguji *service* pada AngularJs yang digunakan untuk mengakses ke *web service*. Pada pengujian ini dilakukan dengan menggunakan Jasmine yang dapat dilihat pada tabel 6.29. Kasus uji pada pengujian ini dapat dilihat pada tabel 6.28, dan hasil dari pengujian dapat dilihat pada gambar 6.25.

Tabel 6.28 Kasus uji *method* GetAvailableEvidenceFactory.get(request) iterasi kedua

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	GetAvailableEvidenceFactory.post(request)	Mengakses URI /TigaPilar/api/ticket/available-evidence dengan HTTP method GET	Mengembalikan *response* dengan *message* Data Tilang dengan Barang Bukti yang belum diambil.	Mengembalikan *response* dengan *message* Data Tilang dengan Barang Bukti yang belum diambil.	Valid

Tabel 6.29 Pengujian *unit method* GetAvailableEvidenceFactory.get(request) iterasi kedua

Pengujian *unit method* GetAvailableEvidenceFactory.get(request)

`it('should return Data Tilang dengan Barang Bukti yang belum diambil', function() {`
`  httpBackend.expect('GET', '/TigaPilar/api/ticket/available-evidence?month=' + params.month + '&requestId=' + UtilService.get() + '&year=' +params.year).respond(200, response);`
`  var result = GetAvailableEvidenceFactory.get(params);`
`  httpBackend.flush();`
`  expect(result.message).toBe("Data Tilang dengan Barang Bukti yang belum diambil");`
`});`

```
Kepolisian - Evidence Service
  should return Data Tilang dengan Barang Bukti yang belum diambil
```

Gambar 6.24 Hasil pengujian *method* GetAvailableEvidenceFactory.get(request) iterasi kedua

6.1.3 Pengujian integrasi

Pengujian integrasi adalah sebuah pengujian dimana menguji komponen-komponen dari perangkat lunak yang telah digabungkan. Pada pengujian integrasi ini dilakukan dengan metode *bottom-up*, yaitu komponen pada tingkat yang lebih rendah diuji terlebuh dahulu sebelum komponen dengan tingkat yang lebih tinggi. Pada pengujian integrasi terdapat yang diuji adalah *method-method* pada *layer controller* dari *web service* yang diimplementasikan.

6.1.3.1 Pengujian integrasi *method* upload(requestId, file)

Pengujian integrasi *method* `upload(requestId, file)` dilakukan dengan kasus uji yang dapat dilihat pada tabel 6.30, kode untuk pengujian dapat dilihat pada tabel 6.31, dan hasil pengujian dapat dilihat pada gambar 6.25.

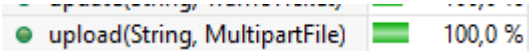
Tabel 6.30 Kasus uji *method* upload(requestId, file)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	upload(requestId, file)	Nilai role sama dengan RoleEnum.KEPOLISIAN dan nilai accessibility sama dengan AccessibilityEnum.CREATE	Memanggil *method* upload(file) dan mengembalikan *response* dengan status ok.	Memanggil *method* upload(file) dan mengembalikan *response* dengan status ok.	Valid
2	upload(requestId, file)	Nilai role selain sama dengan RoleEnum.KEPOLISIAN dan nilai accessibility selain sama dengan AccessibilityEnum.CREATE	Mengembalikan *response* dengan status unauthorized.	Mengembalikan *response* dengan status unauthorized.	Valid

Tabel 6.31 Pengujian integrasi *method* upload(requestId, file)

Pengujian integrasi *method* upload(requestId, file)

```
@Test
public void testUploadSuccess() throws Exception {
    Credential.setRole(RoleEnum.KEPOLISIAN);
    Credential.setAccessibility(AccessibilityEnum.CREATE);
    trafficTicketController.upload(REQUEST_ID, file);
    verify(trafficTicketService, times(1)).upload(file);
}

@Test(expected = UnauthorizedException.class)
public void testUploadFailed() throws Exception {
    Credential.setRole(RoleEnum.KEJAKSAAN);
    Credential.setAccessibility(AccessibilityEnum.READ);
    try {
      trafficTicketController.upload(REQUEST_ID, file);
    } catch (Exception e) {
      throw e;
    }
}
```



Gambar 6.25 Hasil pengujian integrasi *method* upload(requestId, file)

6.1.3.2 Pengujian integrasi *method* sendDataToCourt(requestId)

Pengujian integrasi *method* `sendDataToCourt(requestId)` dilakukan dengan kasus uji yang dapat dilihat pada tabel 6.32, kode pengujian dapat dilihat pada tabel 6.33, dan hasil pengujian dapat dilihat pada gambar 6.26.

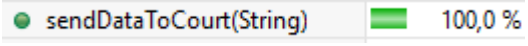
Tabel 6.32 Kasus uji *method* sendDataToCourt(requestId)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	sendDataToCourt (requestId)	Nilai role sama dengan RoleEnum.KEPOLISIAN dan nilai accessibility sama dengan AccessibilityEnum.UPDATE	Memanggil *method* sendDataToCourt (file) dan mengembalikan *response* dengan status ok.	Memanggil *method* sendDataToCourt(file) dan mengembalikan *response* dengan status ok.	Valid
2	sendDataToCourt (requestId)	Nilai role selain sama dengan RoleEnum.KEPOLISIAN dan nilai accessibility selain sama dengan AccessibilityEnum.UPDATE	Mengembalikan *response* dengan status unauthorized.	Mengembalikan *response* dengan status unauthorized.	Valid

Tabel 6.33 Pengujian integrasi *method* sendDataToCourt(requestId)

Pengujian integrasi *method* sendDataToCourt(requestId)

```
@Test
public void testSendDataToCourtSuccess() throws Exception {
  Credential.setRole(RoleEnum.KEPOLISIAN);
  Credential.setAccessibility(AccessibilityEnum.UPDATE);
  mockMvc.perform(put(ControllerPath.TICKET +
  SEND).param("requestId", REQUEST_ID)).andExpect(status().isOk());
  verify(trafficTicketService, times(1)).sendDataToCourt();
}

@Test
public void testSendDataToCourtFailed() throws Exception {
  Credential.setRole(RoleEnum.ADMINISTRATOR);
  Credential.setAccessibility(AccessibilityEnum.CREATE);
  mockMvc.perform(put(ControllerPath.TICKET +
  SEND).param("requestId", REQUEST_ID))
  .andExpect(status().isUnauthorized());
}
```



Gambar 6.26 Hasil pengujian integrasi *method* sendDataToCourt(requestId)

6.1.3.3 Pengujian integrasi *method* save(requestId, trialConfigurationDto)

Pengujian integrasi *method* `save(requestId, trialConfigurationDto)` dilakukan dengan kasus uji yang dapat dilihat pada tabel 6.34, kode pengujian dapat dilihat pada tabel 6.35, dan hasil pengujian dapat dilihat pada gambar 6.27.

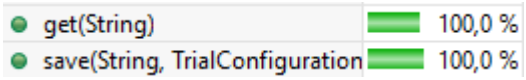
Tabel 6.34 Kasus uji *method* save(requestId, trialConfigurationDto)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	save(requestId, trialonfigurationDto)	Nilai role sama dengan RoleEnum.PENGADILAN dan nilai accessibility sama dengan AccessibilityEnum.CREATE	Memanggil *method* save(trialConfigurationDto) dan mengembalikan *response* dengan status ok.	Memanggil *method* save(trialConfigurationDto) dan mengembalikan *response* dengan status ok.	Valid
2	save(requestId, trialonfigurationDto)	Nilai role selain sama dengan RoleEnum.PENGADILAN dan nilai accessibility selain sama dengan AccessibilityEnum.CREATE	Mengembalikan *response* dengan status unauthorized.	Mengembalikan *response* dengan status unauthorized.	Valid

Tabel 6.35 Pengujian integrasi *method* save(requestId, trialConfigurationDto)

Pengujian integrasi *method* save(requestId, trialConfigurationDto)

```
@Test
public void testSaveSuccess() throws Exception {
  Credential.setRole(RoleEnum.PENGADILAN);
  Credential.setAccessibility(AccessibilityEnum.CREATE);
      mockMvc.perform(post(ControllerPath.TRIAL).param("requestId",
  REQUEST_ID)
  .content(mapper.writeValueAsString(trialConfigurationDto))
  .contentType(MediaType.APPLICATION_JSON_VALUE)).andExpect(status().
  isOk());
  verify(trialConfigurationService,
  times(1)).save(trialConfigurationDto);
}

@Test
public void testSaveFailed() throws Exception {
  Credential.setRole(RoleEnum.KEJAKSAAN);
  Credential.setAccessibility(AccessibilityEnum.READ);
      mockMvc.perform(post(ControllerPath.TRIAL).param("requestId",
  REQUEST_ID).content(mapper.writeValueAsString(trialConfigurationDto
  )).contentType(MediaType.APPLICATION_JSON_VALUE)).andExpect(status(
  ).isUnauthorized());
}
```



Gambar 6.27 Hasil pengujian integrasi *method* save(requestId, trialConfigurationDto)

6.1.3.4 Pengujian integrasi *method* inputCost(requestId, ticketRegisterNumber, cost)

Pengujian integrasi *method* `inputCost(requestId, ticketRgisterNumber, cost)` dilakukan dengan kasus uji yang dapat dilihat pada tabel 6.36, kode pengujian dapat dilihat pada tabel 6.37, dan hasil pengujian dapat dilihat pada gambar 6.28.

Tabel 6.36 Kasus uji *method* inputCost(requestId, ticketRegisterNumber, cost)

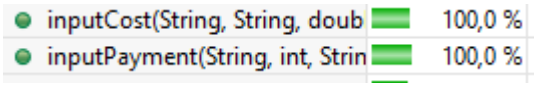
No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	inputCost(requestId, ticketRegisterNumber, cost)	Nilai role sama dengan RoleEnum.PENGADILAN dan nilai accessibility sama dengan AccessibilityEnum.CREATE	Memanggil *method* inputCost(ticketRegisterNumber, cost) dan mengembalikan *response* dengan status ok.	Memanggil *method* inputCost(ticketRegisterNumber, cost) dan mengembalikan *response* dengan status ok.	Valid
2	inputCost(requestId, ticketRegisterNumber, cost)	Nilai role selain sama dengan RoleEnum.PENGADILAN dan nilai accessibility selain sama dengan AccessibilityEnum.CREATE	Mengembalikan *response* dengan status unauthorized.	Mengembalikan *response* dengan status unauthorized.	Valid

Tabel 6.37 Pengujian integrasi *method* inputCost(requestId, ticketRegisterNumber, cost)

Pengujian integrasi *method* inputCost(requestId, ticketRegisterNumber, cost)

```
@Test
public void testInputCostSuccess() throws Exception {
  Credential.setRole(RoleEnum.PENGADILAN);
  Credential.setAccessibility(AccessibilityEnum.CREATE);
  mockMvc.perform(post(ControllerPath.TICKET +
  INPUT_COST).param("requestId", REQUEST_ID)
  .param("ticketRegisterNumber", "A1").param("cost", "20000")
  .contentType(MediaType.APPLICATION_JSON_VALUE)).andExpect(status().
  isOk());
  verify(trafficTicketService, times(1)).inputCost("A1", 20000);
}

@Test
public void testInputCostFailed() throws Exception {
  Credential.setRole(RoleEnum.ADMINISTRATOR);
```

```
    Credential.setAccessibility(AccessibilityEnum.READ);
    mockMvc.perform(post(ControllerPath.TICKET +
    INPUT_COST).param("requestId", REQUEST_ID)
    .param("ticketRegisterNumber", "A1").param("cost",
    "20000").contentType(MediaType.APPLICATION_JSON_VALUE)).andExpect(s
    tatus().isUnauthorized());
  }
```



Gambar 6.28 Hasil pengujian integrasi *method* inputCost(requestId, ticketRegisterNumber, cost)

6.1.3.5 Pengujian integrasi *method* inputPayment(requestId, year, ticketRegisterNumber, paymentDto)

Pengujian integrasi *method* `inputPayment(requestId, year, ticketRgisterNumber, paymentDto)` dilakukan dengan kasus uji yang dapat dilihat pada tabel 6.38, kode pengujian dapat dilihat pada tabel 6.39, dan hasil pengujian dapat dilihat pada gambar 6.29.

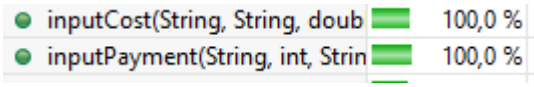
Tabel 6.38 Kasus uji *method* inputCost(requestId, year, ticketRegisterNumber, cost)

No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	inputPayment(requestId, year, ticketRegisterNumber, paymentDto)	Nilai role sama dengan RoleEnum.KEJAKSAAN dan nilai accessibility sama dengan AccessibilityEnum.UPDATE	Memanggil *method* inputPayment(year, ticketRegisterNumber, paymentDto) dan mengembalikan *response* dengan status ok.	Memanggil *method* inputPayment(year, ticketRegisterNumber, paymentDto) dan mengembalikan *response* dengan status ok.	Valid
2	inputPayment(requestId, year, ticketRegisterNumber, paymentDto)	Nilai role selain sama dengan RoleEnum.KEJAKSAAN dan nilai accessibility selain sama dengan AccessibilityEnum.UPDATE	Mengembalikan *response* dengan status unauthorized.	Mengembalikan *response* dengan status unauthorized.	Valid

Tabel 6.39 Pengujian integrasi *method* inputPayment(requestId, year, ticketRegisterNumber, paymentDto)

Pengujian integrasi *method* inputPayment(requestId, year, ticketRegisterNumber, paymentDto)

`@Test`
`public void testInputPaymentSuccess() throws Exception {`
`Credential.setRole(RoleEnum.KEJAKSAAN);`
`Credential.setAccessibility(AccessibilityEnum.UPDATE);`
`mockMvc.perform(put(ControllerPath.TICKET + "/{year}" +`
`"/{ticketRegisterNumber}", 2017, "A1").param("requestId",`
`REQUEST_ID).content(mapper.writeValueAsString(paymentDto))`
`.contentType(MediaType.APPLICATION_JSON_VALUE)).andExpect(status().`
`isOk());`
`verify(trafficTicketService, times(1)).inputPayment(2017, "A1",`
`paymentDto);`
`}`
 	
`@Test`
`public void testInputPaymentFailed() throws Exception {`
`Credential.setRole(RoleEnum.PENGADILAN);`
`Credential.setAccessibility(AccessibilityEnum.CREATE);`
`mockMvc.perform(put(ControllerPath.TICKET + "/{year}" +`
`"/{ticketRegisterNumber}", 2017, "A1").param("requestId",`
`REQUEST_ID).content(mapper.writeValueAsString(paymentDto)).contentT`
`ype(MediaType.APPLICATION_JSON_VALUE)).andExpect(status().isUnautho`
`rized());`
`}`



Gambar 6.29 Hasil pengujian integrasi *method* inputPayment(requestId, year, ticketRegisterNumber, paymentDto)

6.1.3.6 Pengujian integrasi *method* findAvailableEvidence(requestId, year, month) iterasi kedua

Pengujian integrasi *method* `findAvailableEvidence(requestId, year, month)` dilakukan pada iterasi kedua dengan kasus uji yang dapat dilihat pada tabel 6.40, kode pengujian dapat dilihat pada tabel 6.41, dan hasil pengujian dapat dilihat pada gambar 6.30.

Tabel 6.40 Kasus uji *method* findAvailabeEvidence(requestId, year, month) iterasi kedua

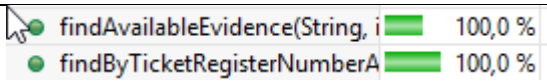
No	Method	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	findAvailableEvidence(requestId, year, month)	Nilai role sama dengan RoleEnum.KEPOLISIAN dan nilai accessibility sama dengan AccessibilityEnum.READ	Memanggil *method* findAvailableEvidence(year, month) dan mengembalikan *response* dengan status ok.	Memanggil *method* findAvailableEvidence(year, month) dan mengembalikan *response*	Valid

				dengan status ok.	
2	findAvailableEvidence(requestId, year, month)	Nilai role selain sama dengan RoleEnum.KEPOLISIAN dan nilai accessibility selain sama dengan AccessibilityEnum.READ	Mengembalikan *response* dengan status unauthorized.	Mengembalikan *response* dengan status unauthorized.	Valid

Tabel 6.41 Pengujian integrasi *method* findAvailableEvidence(requestId, year, month) iterasi kedua

Pengujian integrasi *method* findAvailableEvidence(requestId, year, month)

```
@Test
 public void testFindAvailableEvidenceSuccess() throws Exception {
  Credential.setRole(RoleEnum.KEPOLISIAN);
  Credential.setAccessibility(AccessibilityEnum.READ);
  mockMvc.perform(get(ControllerPath.TICKET +
  AVAILABLE_EVIDENCE).param("requestId", REQUEST_ID).param("year",
  "0").param("month", "0")).andExpect(status().isOk());
  verify(trafficTicketService, times(1)).findAvailableEvidence(0, 0);
}

@Test
 public void testFindAvailableEvidenceFailed() throws Exception {
  Credential.setRole(RoleEnum.PENGADILAN);
  Credential.setAccessibility(AccessibilityEnum.CREATE);
  mockMvc.perform(get(ControllerPath.TICKET +
  AVAILABLE_EVIDENCE).param("requestId", REQUEST_ID).param("year",
  "0").param("month", "0")).andExpect(status().isUnauthorized());
}
```



Gambar 6.30 Hasil pengujian integrasi *method* findAvailableEvidence(requestId, year, month) iterasi kedua

6.2 Pengujian Non Fungsional

Pengujian non fungsional dilakukan untuk menguji kebutuhan non fungsional yang telah didefinisikan pada tahap analisis. Pada pengujian non fungsional terdapat dua pengujian yaitu pengujian *usability*, dan pengujian *data integrity*. Pengujian *usability* digunakan untuk melihat seberapa tingkat kemudahan sistem yang telah dibuat. Sedangkan pengujian *data integrity* untuk memastikan bahwa data hanya dapat diakses oleh pengguna yang memiliki wewenang sesuai dengan tugasnya.

6.2.1 *Usability Testing*

Usability testing digunakan untuk melihat tingkat kepuasaan pengguna terhadap sistem yang dibuat. Pengujian dilakukan oleh 5 orang pengguna yaitu : seorang administrator, operator kepolisian, operator pengadilan, operator

kejaksaan, dan masyarakat umum. Setelah pengguna menjalankan aplikasi dengan diberikan beberapa *task* yang harus diselesaikan, isi *task* dapat dilihat pada tabel 6.42. kemudian dilakukan pengujian dengan memberikan kuisioner kepada responden. Kuisioner terdiri dari 10 pertanyaan. Isi dan hasil dari kuisioner dapat dilihat padal lampiran C.

Tabel 6.42 Daftar *task* pengujian *usability*

Pengguna	No	Nama Task	Task
Administrator	1	A01	Melakukan autentikasi ke sistem
	2	A02	Mengakses halaman data tilang
	3	A03	Mencari nama “Anton”
	4	A04	Melakukan edit data tilang, mengubah denda tilang sebesar 90000
Operator Kepolisian	1	K01	Melakukan autentikasi ke sistem
	2	K02	Mengakses halaman *upload*
	3	K03	Mengunggah *file* data tilang
	4	K04	Mencari nama citra
	5	K05	Mengirimkan *file* data tilang
Operator Pengadilan	1	P01	Melakukan autentikasi ke sistem
	2	P02	Mengakses ke halaman data tilang minggu ini
	3	P03	Menambahkan konfigurasi persidangan
	4	P04	Mencari nama “Budi”
	5	P05	Memasukkan denda tilang 120000 pada nama “Budi”
Operator Kejaksaan	1	KJ01	Melakukan autentikasi ke sistem
	2	KJ02	Mengakses halaman data tilang
	3	KJ03	Mencetak laporan data tilang dengan tanggal putusan 6 Desember 2017
	4	KJ04	Mencari nama budi
	5	KJ05	Melakukan konfirmasi pembayaran di bank dengan nama pelanggar budi dengan tanggal bayar 28 November 2017
Masyarakat	1	M01	Mengakses *web* Tiga Pilar
	2	M02	Melakukan *filtering* data tilang dengan tanggal 7 desember 2017
	3	M03	Mencari nama “Anton”

Setelah melakukan pengujian, didapatkan hasil pengujian berupa nilai dari *effectiveness* dan SUS dari *task* dan kuisioner yang telah diberikan. Hasil pengujian *effectiveness* dapat dilihat pada tabel 6.43 dan hasil pengujian SUS dapat dilihat

pada tabel 6.44. Dalam tabel tersebut dijelaskan nilai yang dihasilkan oleh masing-masing pengguna berdasarkan data pengujian.

Tabel 6.43 Hasil pengujian *usability effectiveness*

Aktor	***Task***				
Administrator	A01	A02	A03	A04	
	1	1	1	1	
Operator Kepolisian	K01	K02	K03	K04	K05
	1	1	0	1	1
Operator Pengadilan	P01	P02	P03	P04	P05
	1	1	1	1	1
Operator Kejaksaan	KJ01	KJ02	KJ03	KJ04	KJ05
	1	1	1	1	0
Masyarakat	M01	M02	M03		
	1	1	1		

Tabel 6.44 Nilai akhir pengujian *usability*

No	**Nama**	**Nilai SUS**	**Keterangan**
1	Narendra Wahyu Widiatmoko (Administrator)	72,5	B
2	Gunawan (Operator Kepolisian)	60	C
3	Prayogo Hadi Purwanto (Operator Pengadilan)	80	A
4	Adhityo Puji Wiyono (Operator Kejaksaan)	67,5	C
5	Melinda Sri Wulandari (Masyarakat umum)	82,5	A

Nilai rata-rata SUS yang di dapatkan dari dari keseluruhan pengujian adalah 72,5 dan nilai *effectiveness* adalah 90,9%. Hasil dari nilai SUS ini masih dibatas ambang bawah meskipun diatas dari rata-rata sistem yang dapat dikatakan mudah digunakan, hal ini dikarenakan waktu yang disediakan untuk pengguna mempelajari sistem relatif cepat yaitu kurang lebih sekitar 30 menit.

6.2.2 Pengujian *data integrity*

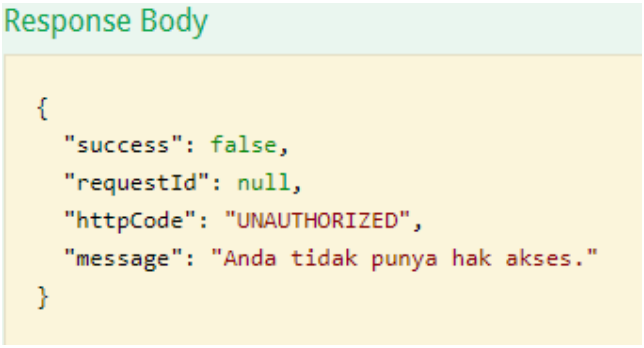
Pengujian *data integrity* digunakan untuk melakukan pengujian bahwa sistem hanya boleh diakses oleh *user* yang memiliki wewenang sesuai dengan tugasnya. Pada pengujian *data integrity* dilakukan dengan cara mengakses ke *web service* yang telah dibuat melalui swagger.

6.2.2.1 Pengujian *web service* mengunggah data tilang

Pengujian *web service* mengunggah data tilang dengan kasus uji yang dapat dilihat pada tabel 6.45. Hasil dari pengujian ini dapat dilihat pada gambar 6.31.

Tabel 6.45 Kasus uji *web service* mengunggah data tilang

No	URI *web service*	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	/TigaPilar/api/upload	Mengakses sebagai operator pengadilan.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Valid



Gambar 6.31 Hasil pengujian *web service login*

6.2.2.2 Pengujian *web service* mengirimkan data tilang

Pengujian *web service* mengirimkan data tilang dengan kasus uji sebagai operator pengadilan. Kasus uji dapat dilihat pada tabel 6.46. Hasil yang diharapkan dari pengujian ini adalah sistem akan memberikan *response* dengan *message* Anda tidak punya hak aksses yang dapat dilihat pada gambar 6.32.

Tabel 6.46 Kasus uji *web service* mengirimkan data tilang

No	URI *web service*	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	/TigaPilar/api/ticket/send	Mengakses sebagai operator pengadilan.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Valid

Response Body


```
{
  "success": false,
  "requestId": null,
  "httpCode": "UNAUTHORIZED",
  "message": "Anda tidak punya hak akses."
}
```


Gambar 6.32 Hasil pengujian *web service* mengirimkan data tilang

6.2.2.3 Pengujian *web service* menambahkan konfigurasi persidangan

Pengujian *web service* menambahkan konfigurasi persidangan dengan kasus uji sebagai operator kepolisian. Kasus uji dapat dilihat pada tabel 6.47. Hasil yang diharapkan dari pengujian ini adalah sistem akan memberikan *response* dengan *message* Anda tidak punya hak aksses yang dapat dilihat pada gambar 6.33.

Tabel 6.47 Kasus uji *web service* menambahkan konfigurasi persidangan

No	URI *web service*	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	/TigaPilar/api/trial	Mengakses sebagai operator kepolisian.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Valid

Response Body


```
{
  "success": false,
  "requestId": null,
  "httpCode": "UNAUTHORIZED",
  "message": "Anda tidak punya hak akses."
}
```


Gambar 6.33 Hasil pengujian *web service* menambahkan konfigurasi persidangan

6.2.2.4 Pengujian *web service* memasukkan denda tilang

Pengujian *web service* memasukkan denda tilang dengan kasus uji sebagai operator kepolisian. Kasus uji dapat dilihat pada tabel 6.48. Hasil yang diharapkan dari pengujian ini adalah sistem akan memberikan *response* dengan *message* Anda tidak punya hak aksses yang dapat dilihat pada gambar 6.34.

Tabel 6.48 Kasus uji *web service* memasukkan denda tilang

No	URI *web service*	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	/TigaPilar/api/ticket/inpu-cost	Mengakses sebagai operator kepolisian.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Valid

Response Body


```
{
  "success": false,
  "requestId": null,
  "httpCode": "UNAUTHORIZED",
  "message": "Anda tidak punya hak akses."
}
```

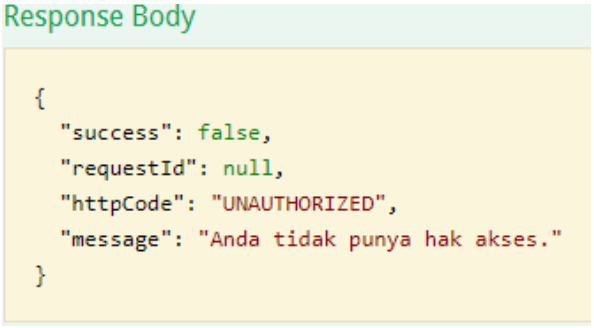

Gambar 6.34 Hasil pengujian *web service* memasukkan denda tilang

6.2.2.5 Pengujian *web service* konfirmasi pembayaran tilang

Pengujian *web service* konfirmasi pembayaran dengan kasus uji sebagai operator kepolisian. Kasus uji dapat dilihat pada tabel 6.49. Hasil yang diharapkan dari pengujian ini adalah sistem akan memberikan *response* dengan *message* Anda tidak punya hak aksses yang dapat dilihat pada gambar 6.35.

Tabel 6.49 Kasus uji *web service* konfirmasi pembayaran tilang

No	URI *web service*	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	/TigaPilar/api/ticket/{year}/{ticketRegisterNumber}	Mengakses sebagai operator kepolisian.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Valid



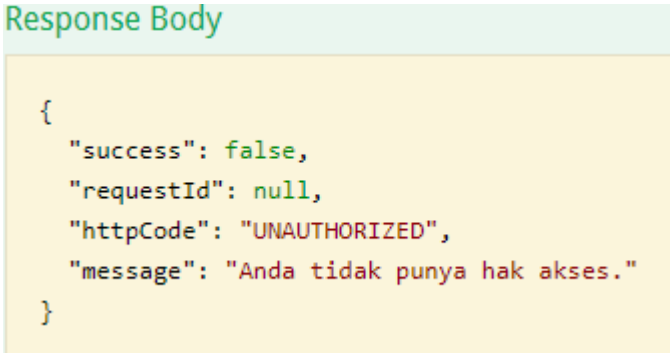
Gambar 6.35 Hasil pengujian *web service* konfirmasi pembayaran tilang

6.2.2.6 Pengujian *web service* menampilkan barang bukti yang belum diambil (iterasi kedua)

Pengujian *web service* menampilkan barang bukti yang belum diambil dilakukan pada iterasi kedua dengan kasus uji sebagai operator pengadilan. Kasus uji dapat dilihat pada tabel 6.50. Hasil yang diharapkan dari pengujian ini adalah sistem akan memberikan *response* dengan *message* Anda tidak punya hak aksses yang dapat dilihat pada gambar 6.36.

Tabel 6.50 Kasus uji *web service* menampilkan barang bukti yang belum diambil (iterasi kedua)

No	URI *web service*	Kasus Uji	Hasil yang diharapkan	Hasil	Status
1	/TigaPilar/api/ticket/available-evidence	Mengakses sebagai operator pengadilan.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Memberikan *response* dengan *message* Anda tidak punya hak akses.	Valid



Gambar 6.36 Hasil pengujian *web service* menampilkan barang bukti yang belum diambil (iterasi kedua)